Assoziative Arrays

Beispiel

Wir wollen die Hauptstädte einiger Länder Speichern, d.h. wir benötigen eine Zuordnung wie

```
Hauptstadt von Frankreich = Paris
```

Es verhält sich also ebenso wie bei gewöhnlichen Arrays, wir wollen viele gleichartige Objekte Speichern (Hauptstädte). Allerdings wäre es in diesem fall günstig, wenn wir bei der Art der Speicherung direkt einen Zusammenhang zum Land herstellen könnten. Ein Ausdruck in der Art `hauptstadt[Frankreich]=Paris` wäre viel besser als `hauptstadt[0] = Paris`, da dieser einen direkten Zusammenhang zwischen dem Land und seiner Hauptstadt herstellt.

Definition

Ein **assoziatives Array** (auch Dictionary, Liste von Schlüssel-Wert-Paaren oder assoziatives Datenfeld) ist eine Datenstruktur, bei der anders als bei einem gewöhnlichen Array auch nichtnumerische (oder nicht fortlaufende) Schlüssel, zumeist Zeichenketten, verwendet werden können, um die enthaltenen Elemente zu adressieren. Die Elemente sind nicht in einer festgelegten Reihenfolge abgespeichert. Idealerweise werden die Schlüssel so gewählt, dass eine für die Programmierer nachvollziehbare Verbindung zwischen Schlüssel und Datenwert besteht. Die meisten Programmiersprachen unterstützen assoziative Arrays - Java auch.

Benutzung in Java

Assoziative Arrays werden in Java durch die "HaspMap"-Klasse zur Verfügung gestellt. Um diese benutzen zu können, muss man zunächst die zugehörige Bibliothek einbinden:

```
import java.util.HashMap; // importiere die HashMap Klasse
```

dann kann man im Programmcode ein assoziatives Array folgendermaßen definieren:

```
// Assoziatives Array, die Keys sind vom Typ String, die Werte sind vom Typ String
// das ist für unser Hauptstadtbeispiel sinnvoll.
HashMap<String, String> hauptstadt = new HashMap<String, String>();
```

Die vollständige Klassendokumentation findest du hier.

Wichtige Methoden bei der Verwendung von HashMaps

Elemente hinzufügen

Um einer HashMap ein Element hinzuzufügen, verwendet man die `put()`-Methode:

```
[...]
import java.util.HashMap;
[...]

[...]
// Erzeuge eine HashMap Objekt
HashMap<String, String> hauptstadt = new HashMap<String, String>();
[...]

[...]
// Schlüssel/Wert-Paare hinzufügen (Country, City)
hauptstadt.put("England", "London");  // "hauptstadt von England = London"
hauptstadt.put("Germany", "Berlin");
hauptstadt.put("Norway", "Oslo");
hauptstadt.put("USA", "Washington DC");
System.out.println(capitalCities);
[...]
```

Ein Element auslesen

Um den Wert eines Elements einer HashMap auszulesen verwendet man die get()-Methode mit dem Schlüssel des Elements:

```
String hsengland = hauptstadt.get("England");
```

Elemente löschen

Elemente löschen kann man unter Angabe des Schlüssels mit der Methode remove().

```
hauptstadt.remove("England");
```

Um **alle** Elemente zu löschen, kann man die Methode clear() verwenden:

```
hauptstadt.clear();
```

Anzahl der Elemente herausfinden

Mit Hilfe der Methode size() kann man die Zahl der Elemene in den HashMap herausfinden:

```
int elementAnzahl = hauptstadt.size();
```

Operationen auf allen Elementen: Eine HashMap mit einer Schleife durchlaufen

Die Elemente einer HashMap kann man mit einer foreach-Schleife durchlaufen. Dabei hat man im wesentlichen zwei Möglichkeiten:

(1) Man durchläuft die Menge aller Schlüssel und beschafft sich die Werte zu den Schlüsseln mit der get()-Methode

Die Menge aller Schlüssel erhält man mit der Methode keySet().

```
// Gibt die Schlüssel aus
for (String key : hauptstadt.keySet()) {
  System.out.println(key);
}

// Gibt Schlüssel und Werte aus
for (String key : hauptstadt.keySet()) {
  System.out.println("key: " + key + " value: " + hauptstadt.get(key));
}
```

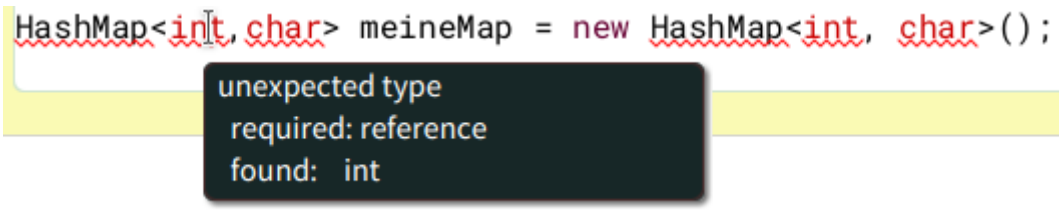
(2) Wenn man an den Schlüsseln nicht interessiert ist, kann man direkt die Werte durchlaufen

Mit der Methode values() kann man sich direkt die Menge der Werte beschaffen, und diese mit einer foreach-Schleife durchlaufen.

```
// Nur die Werte beschaffen und ausgeben
for (String stadt : hauptstadt.values()) {
  System.out.println(stadt);
}
```

Anmerkung zu Wrapper-Klassen

Schlüssel und Werte bei HashMaps sind stets **Objekte**. Aus diesem Grund schlägt eine Verwendung mit primitiven Datentypen wie int, char , double fehl: `HashMap<int,char> meineMap = new HashMap<int, char>();` liefert eine Fehlermeldung.



Das Problem wird gelöst, indem man für die primitiven Datentypen die zugehörigen Wrapper-Klassen verwendet:

```
HashMap<Integer,Character> meineMap = new HashMap<Integer, Character>();
```

Übersicht über die Wrapper-Klassen

Primitiver Typ	Wrapper Klasse
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double
char	Character
boolean	Boolean